examples from calc 1

Ben Bolker

03 October 2019

## Tuples

* simple; **non-mutable** version of lists
* faster, safer
* can be expressed as x, y, z (or (x,y,z), probably clearer)
* empty tuple: ()
* tuple with one element: (x,)
* can do many of the same things as with lists

x = (1,4,"a",3)  
print(x[1]) ## indexing

## 4

print(x[2:]) ## slicing

## ('a', 3)

print(x+(3,)) ## appending

## (1, 4, 'a', 3, 3)

print(x[:2] + (3,) + x[2:]) ## insertion in the middle

## (1, 4, 3, 'a', 3)

x.index(4) ## indexing

## 1

"z" in x ## looking for stuff

## False

x.count(4) ## count

## 1

* you *can’t* modify the existing tuple at all (deletion, modification)
* unpacking: x,y,z = t
* swapping: (a,b) = (b,a)
* useful as the return value of functions; safe, and can be unpacked
* convert to/from lists (tuple(), list())

x = (1,2,3)  
def modify(x):  
 y = list(x)  
 y[0] = "a"  
 return(tuple(y))  
  
print(modify(x))

## ('a', 2, 3)

print(x)

## (1, 2, 3)

## reminders/clarifications

* parentheses (()) vs square brackets ([])
* **square brackets**
  + indexing (lists or strings or tuples): x[5]
  + slicing (lists or strings or tuples): x[5:7]
  + defining lists: [1,2,3]
* **parentheses**
  + order of operations: (1+2)\*3, a and (not b or c)
  + calling functions: len(x), range(5), print("hello")
  + calling methods: x.sort(), x.append(4)
  + defining functions: def f(x1,x2,x3):
  + returning values from functions: return(x) (\*)
  + defining tuples: (), (1,), (2,3) (\*)

“*" actually (mostly)* optional\*: see [here](https://stackoverflow.com/questions/4978567/should-a-return-statement-have-parentheses)

## Root-finding methods

* Assume that is a continuous function on the real numbers.
* Suppose that
* Suppose *endpoints are of opposite signs*:  
   and **or** and
* (or )
* By the Intermediate Value Theorem, there is some number between and with ; this is called a **root** of the function

We will use three methods (Grid, Bisection, and Newton’s method) to approximate such a number . (There may be more than one root of in the interval between and .)

## Example

* We’ll use as an example
* impossible to do analytically!
* value at 0 = -3/2
* value at 1 =

## Grid Method

* Break the interval into subintervals of equal sizes, having endpoints
* Compute
* Find the index such that is closest to 0 and use this to approximate a root of in the interval .
* **Project:** Create a function grid\_search(f, a, b, n) that implements the grid method.

## Bisection Method

* Bisect the interval into two equal subintervals , , where .
* If and have opposite signs, then there will be a root in . Otherwise, there will be a root in .
* Bisect this subinterval ( in the former case, in the latter), and continue bisecting until the subinterval is small.
* A root of will be located in this small subinterval.
* **Project:** Create a function bisect(f, a, b, tol) that approximates a root of f in the interval [a, b] with an error of at most tol.

## Newton’s Method

* Suppose we know the derivative (*gradient*) as well as
* For a given starting value , guess the position of the root according to .
* Repeat until we are within tolerance of the root ( is small
* **Project:** Create a function newton(f, grad, x0, tol, nmax) that approximates a root of f with an error of at most tol, taking no more than nmax steps.